浙江大學
ZHEJIANG UNIVERSITY

Computer Architecture Experiment

Topic 8. Cache Design

浙江大学计算机学院
陈文智
chenwz@zju.edu.cn

实验操作流程

- 阅读实验文档，了解Cache的理论内容
- 设计Cache Management Unit和Cache Line。
- 对Cache单元设计测试激励进行仿真，检验Cache单元的仿真结果是否符合要求。

实验验收标准

- 仿真执行过程中，处理器的行为和内部控制信号均符合要求。
- 下载至开发板后的单步执行过程中，寄存器的变化过程和最终执行结果与测试程序相吻合。

Outline

- Experiment Purpose
- Experiment Task
- Basic Principle
- Operating Procedures
- Precaution
- Checkpoints

Experiment Purpose

- Understand Cache Line.
- Understand the principle of Cache Management Unit (CMU) and State Machine of CMU.
- Master the design methods of CMU.
- Master the design methods of Cache Line.
- master verification methods of Cache Line.

Experiment Task

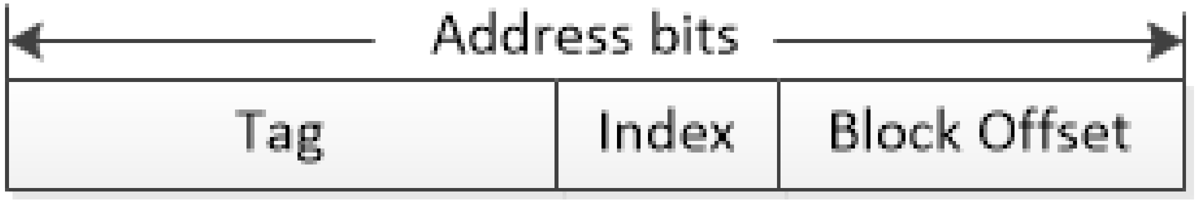
- **Design of Cache Line and CMU.**
- **Verify the Cache Line and CMU.**
- **Observe the Waveform of Simulation.**

Cache Line

V	D	tag	Data

Cache Mode

- **Direct Map**
- **Write Back**
- **Write Allocate**



Address

- Bytes of word = 4, WORD_BYTES_WIDTH = 2
- Words of line = 4, LINE_WORDS_WIDTH = 2
- Tag bits = 22
- Address bits = 32
- LINE_INDEX_WIDTH = ADDR_BITS - TAG_BITS - LINE_WORDS_WIDTH - WORD_BYTES_WIDTH = ?
- Line Number = ?

Cache Line Memory

- reg [LINE_NUM-1:0] inner_valid = 0;
- reg [LINE_NUM-1:0] inner_dirty = 0;
- reg [TAG_BITS-1:0] inner_tag [0:LINE_NUM-1];
- reg [WORD_BITS-1:0] inner_data [0:LINE_NUM*LINE_WORDS-1];

Input and output Signals of Cache Line

```
module cache (
            input wire clk,  // clock
            input wire rst,  // reset
            input wire [ADDR_BITS-1:0] addr,  // address
            input wire store,  // set valid to 1 and reset dirty to 0
            input wire edit,  // set dirty to 1
            input wire invalid,  // reset valid to 0
            input wire [WORD_BITS-1:0] din,  // data write in
            output wire hit,  // hit or not
            output reg [WORD_BITS-1:0] dout,  // data read out
            output reg valid,  // valid bit
            output reg dirty,  // dirty bit
            output reg [TAG_BITS-1:0] tag  // tag bits
            );
```

Read and Write Cache

```
dout <= inner_data[addr[ADDR_BITS-TAG_BITS-1:WORD_BYTES_WIDTH]];


if ( ???|| ???)
   inner_data[addr[ADDR_BITS-TAG_BITS-1:WORD_BYTES_WIDTH]] <= din;
```

Dirty, Valid, tag of Cache

- **invalid**

inner_valid[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=?

inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <= ?

- **store**

inner_valid[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=?

inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=?

inner_tag[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=?

- **edit**

inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=?

inner_tag[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]] <=?

output of Dirty, Valid, tag, hit

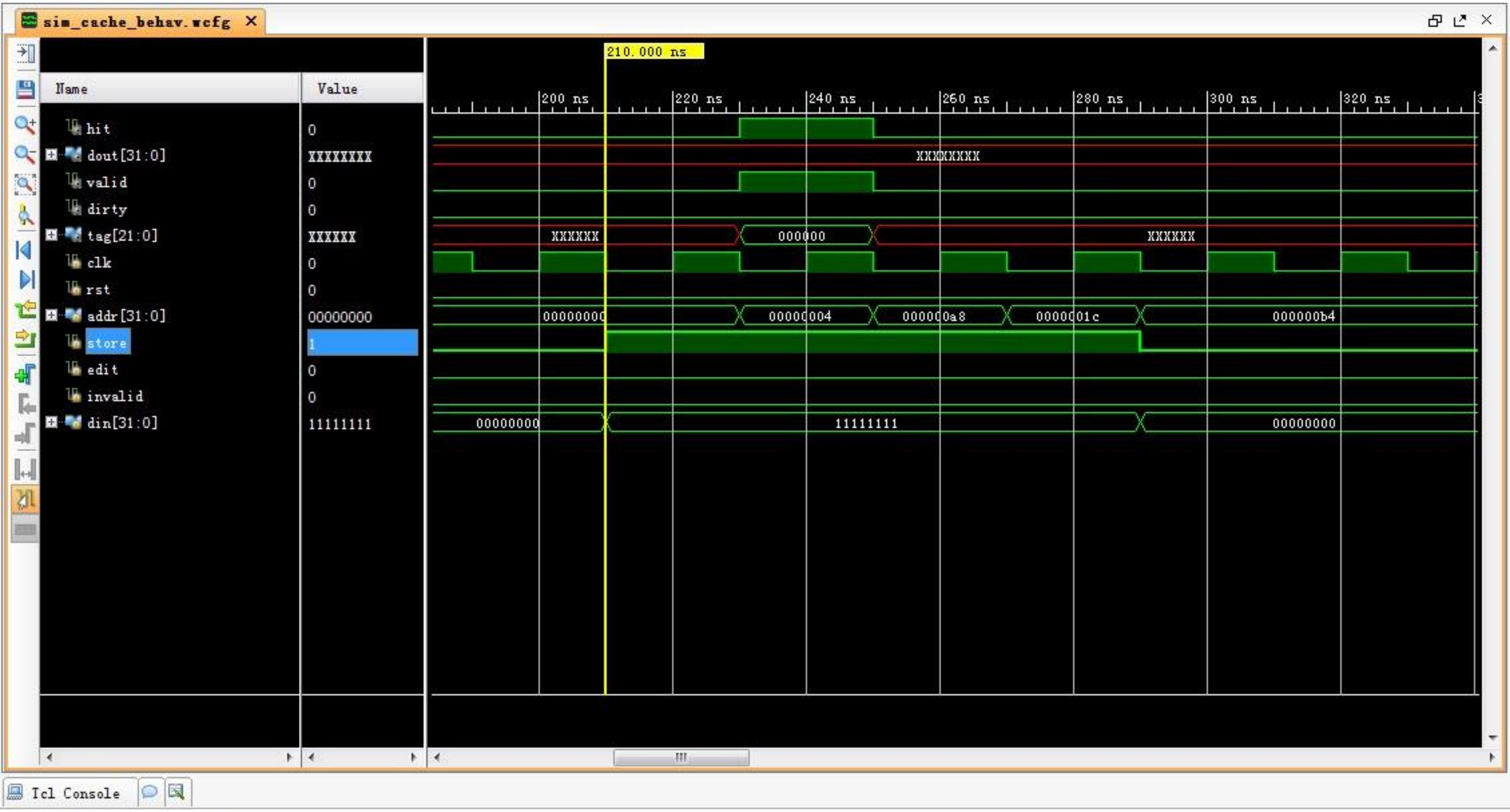
```
valid <= inner_valid[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]]

dirty <= inner_dirty[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]]

tag <= inner_tag[addr[ADDR_BITS-TAG_BITS-1:LINE_WORDS_WIDTH+WORD_BYTES_WIDTH]]
```

hit = ?? & ??

Simulation Example

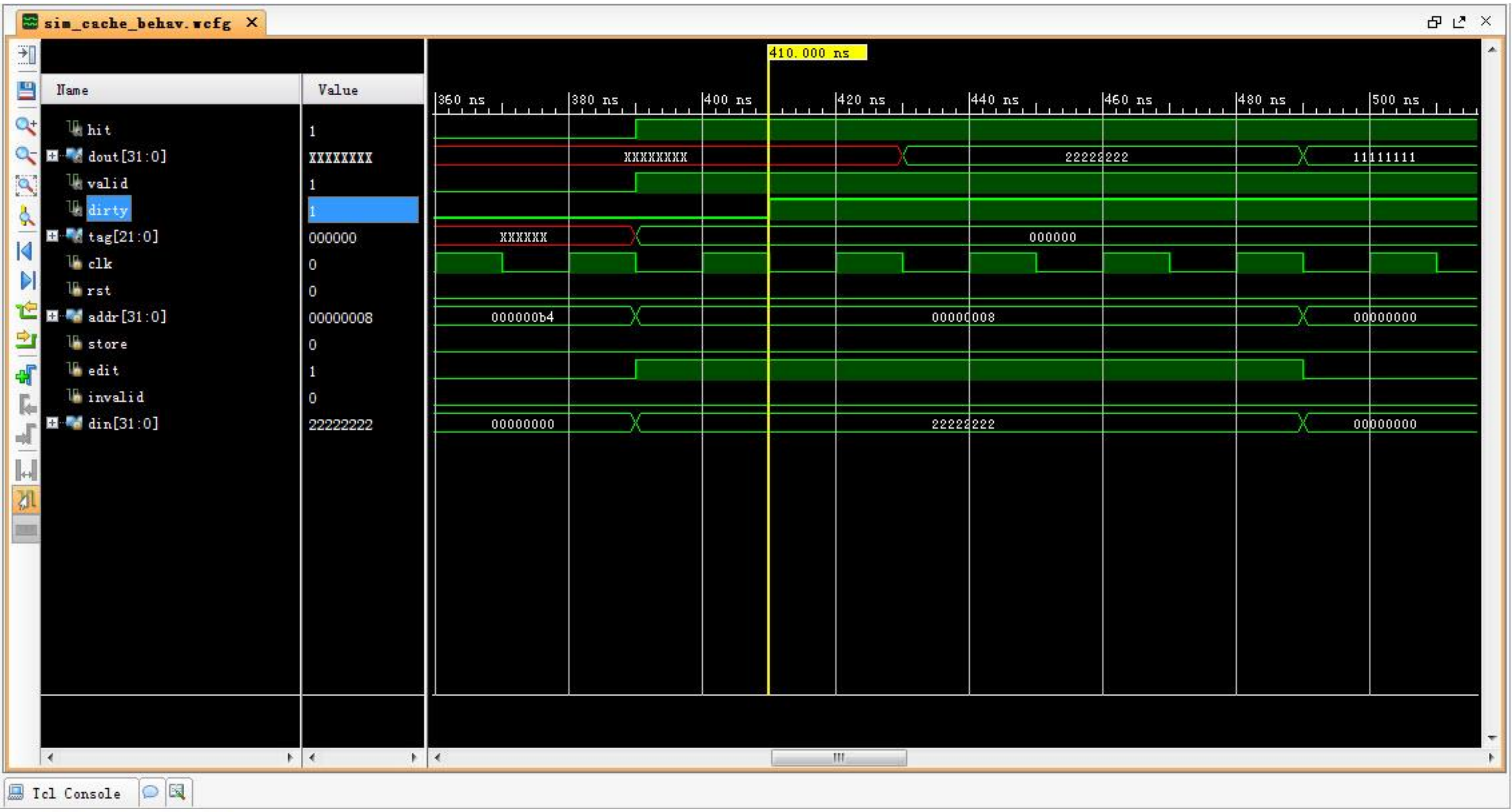
```
clk=1;

#210 store = 1; din = 32'h11111111; addr = 32'h00000000;

#20 addr = 32'h00000004;

#20 addr = 32'h000000A8;

#20 addr = 32'h0000001C;

#20 store = 0; addr = 32'h000000B4; din = 0;

#100  edit = 1; din = 32'h22222222; addr = 32'h00000008;

#100 edit = 0; din = 0; addr = 0;


initial forever #10 clk = ~clk;
```

Simulation Example



Name	Value
hit	0
dout[31:0]	XXXXXXXX
valid	0
dirty	0
tag[21:0]	XXXXXX
clk	0
rst	0
addr[31:0]	00000000
store	1
edit	0
invalid	0
din[31:0]	11111111

Simulation Example

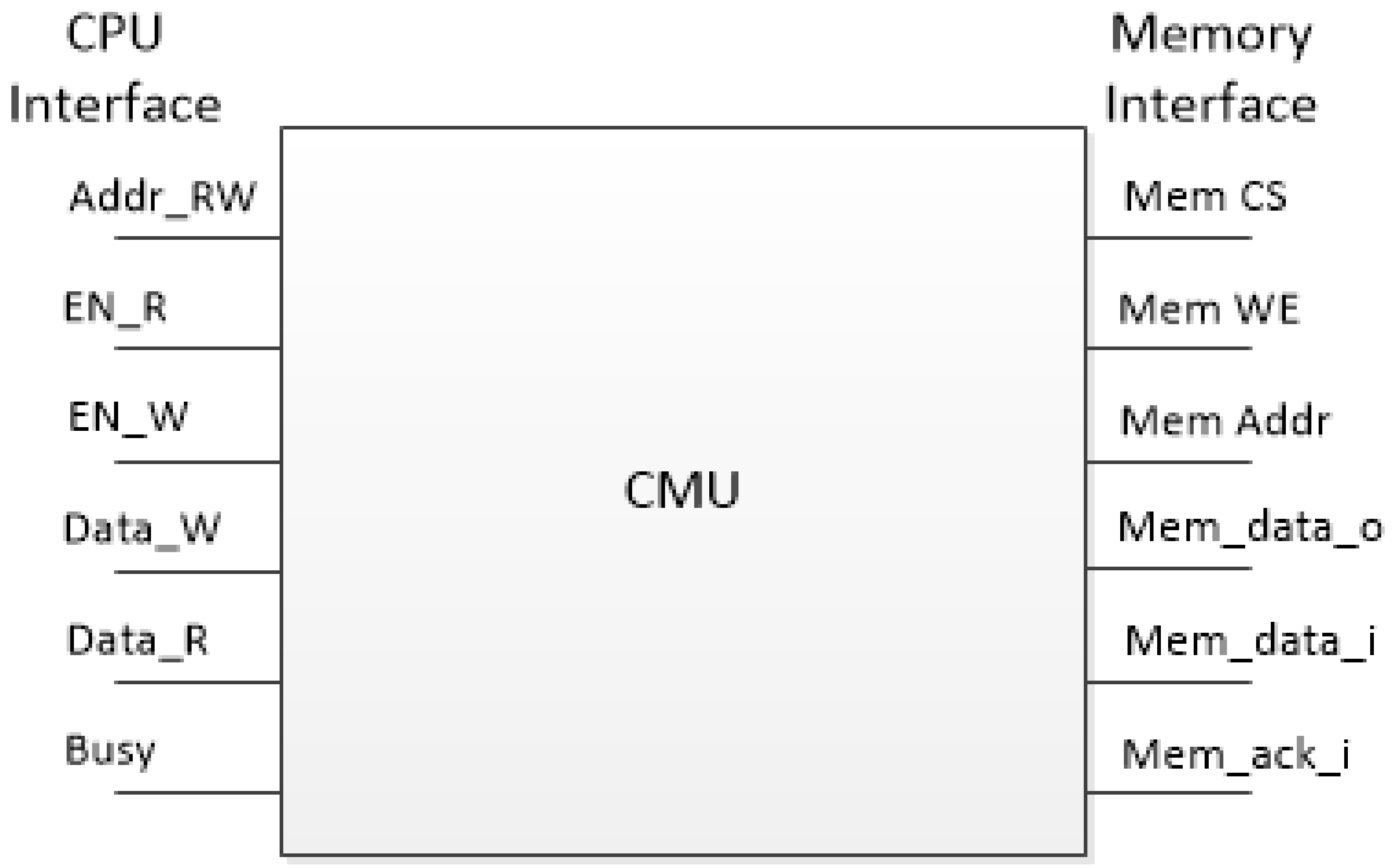


Name	Value
hit	1
dout[31:0]	XXXXXXXX
valid	1
dirty	1
tag[21:0]	000000
clk	0
rst	0
addr[31:0]	00000008
store	0
edit	1
invalid	0
din[31:0]	22222222

Simulation

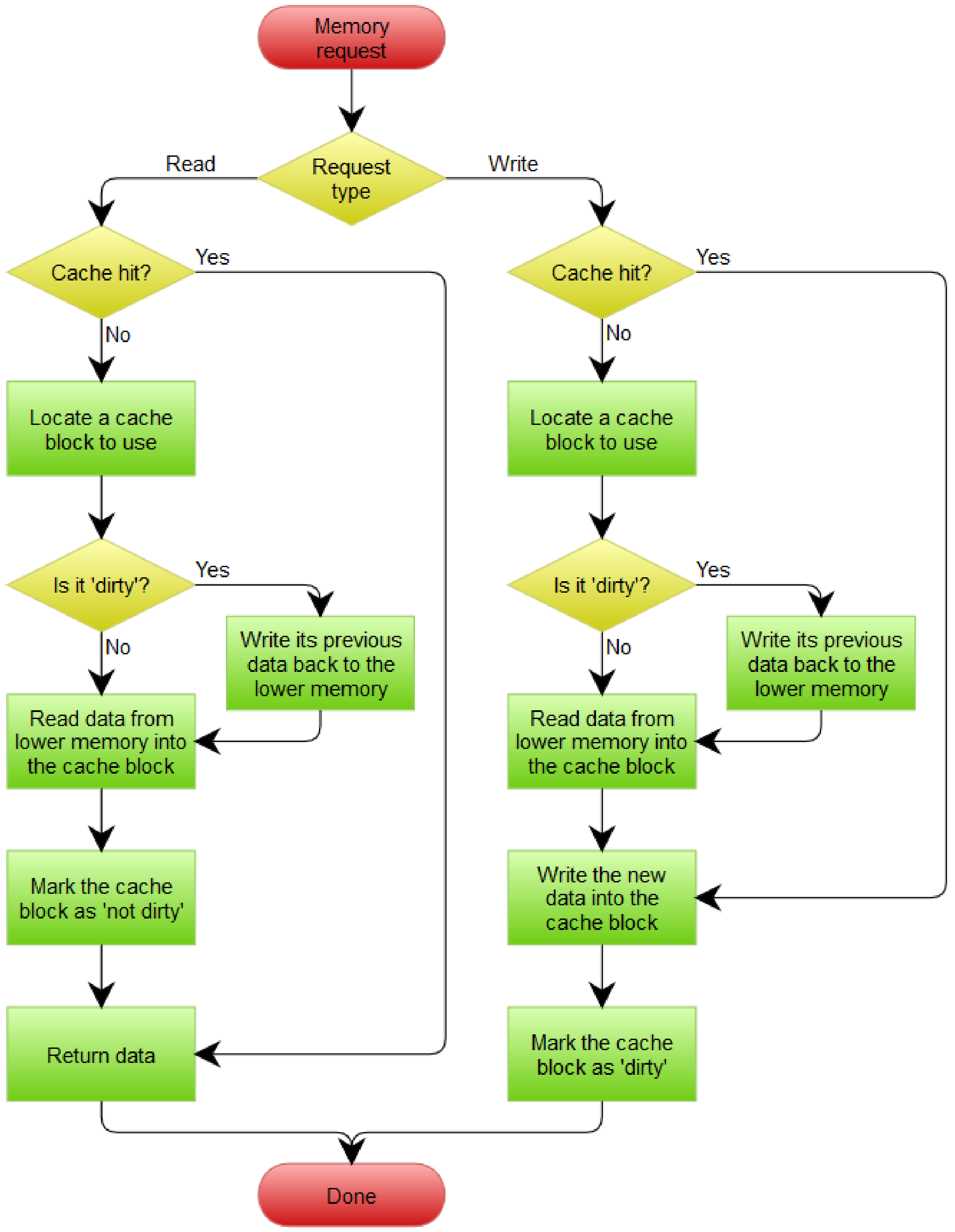
- **Write Simulation Code yourself**
  - cache initialization
  - read
    - **miss**
    - **hit**
  - write
    - **miss**
    - **hit**

Cache Management Unit



Cache Operati

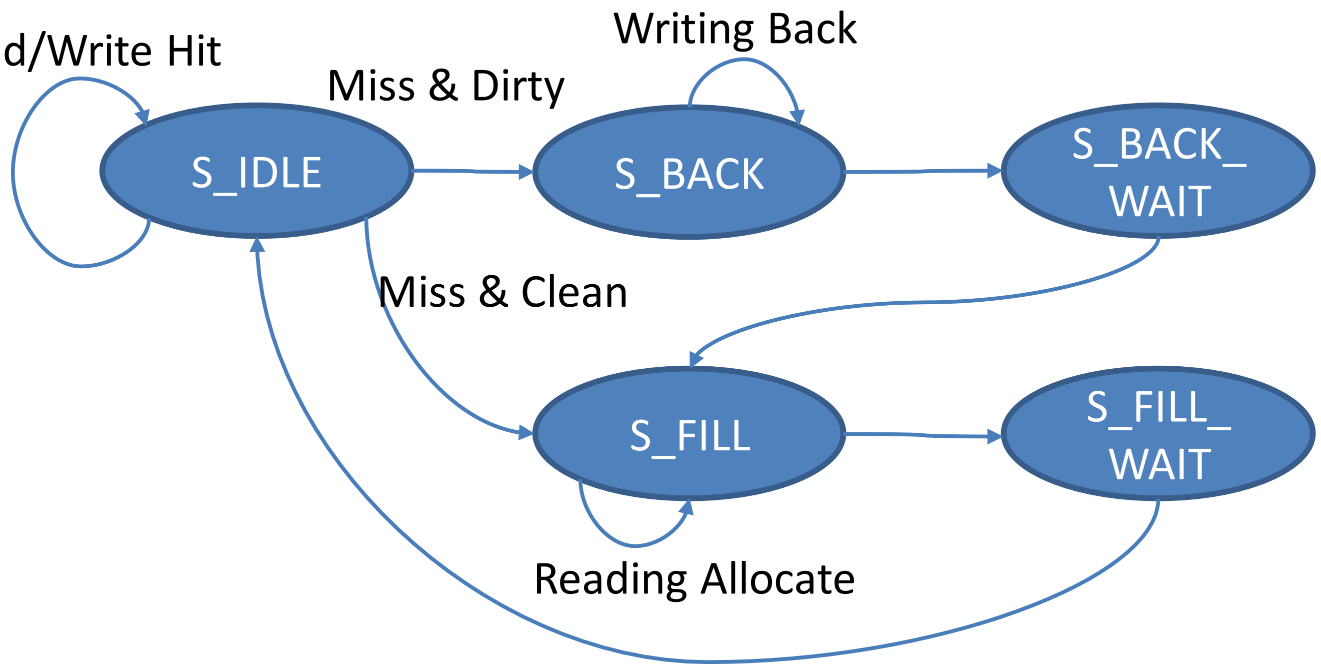
- **Read (Hit/Miss)**
- **Write (Hit/Miss)**
- **Replace (Clean/Dirty)**



Memory request
Request type
Read
Write
Cache hit?
Yes
No
Locate a cache block to use
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Mark the cache block as 'not dirty'
Return data
Cache hit?
Yes
No
Locate a cache block to use
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Write the new data into the cache block
Mark the cache block as 'dirty'
Done

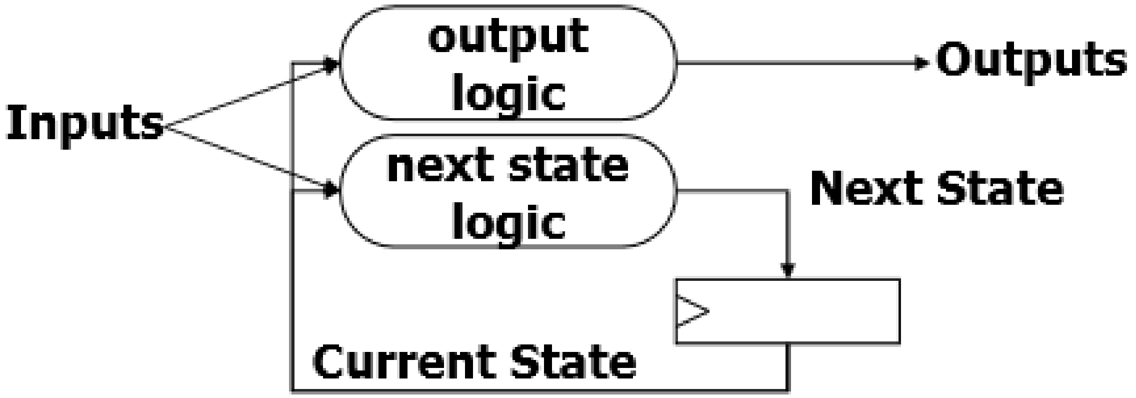
浙江大学
ZheJiang University

Cache Management State Machine



State Machine

- Next State Logic
- State assignment
- Output



Next logic (1)

```
S_IDLE: begin
        if (en_r || en_w) begin
                if (cache_hit)
                        next_state = ???;
                else if (cache_valid && cache_dirty)
                        next_state = ???;
                else
                        next_state = ???;
        end
end
S_BACK: begin
        if (mem_ack_i)
                next_word_count = word_count + 1'h1;
        else
                next_word_count = word_count;
        if (mem_ack_i && word_count == {LINE_WORDS_WIDTH{1'b1}})
                next_state = ???;
        else
                next_state = ???;
end
```

Next logic (2)

```
S_BACK_WAIT: begin
        next_word_count = 0;
        next_state = ???;
end
S_FILL: begin
        if (mem_ack_i)
                next_word_count = word_count + 1'h1;
        else
                next_word_count = word_count;
        if (mem_ack_i && word_count == {LINE_WORDS_WIDTH{1'b1}})
                next_state = ???;
        else
                next_state = ???;
end
S_FILL_WAIT: begin
        next_word_count <= 0;
        next_state <= ???;
end
```

Perform State Assignment

```
always @(posedge clk) begin
        if (rst) begin
                state <= 0;
                word_count <= 0;
        end
        else begin
                state <= next_state;
                word_count <= next_word_count;
        end
end
```

Output (1)

```
case (next_state)
        S_IDLE: begin
                cache_addr = addr_rw;
                cache_edit = en_w;
                cache_din = data_w;
        end
        S_BACK, S_BACK_WAIT: begin
                cache_addr = {addr_rw[31:LINE_WORDS_WIDTH+2], next_word_count,
2'b00};
        end
        S_FILL, S_FILL_WAIT: begin
                cache_addr = {addr_rw[31:LINE_WORDS_WIDTH+2], word_count_buf,
2'b00};
                cache_din = mem_data_syn;
                cache_store = mem_ack_syn;
        end
endcase
```

Output (2)

```
case (next_state)
        S_IDLE, S_BACK_WAIT, S_FILL_WAIT: begin
                mem_cs_o <= 0;
                mem_we_o <= 0;
                mem_addr_o <= 0;
        end
        S_BACK: begin
                mem_cs_o <= 1;
                mem_we_o <= 1;
                mem_addr_o <= {cache_tag, addr_rw[31-
TAG_BITS:LINE_WORDS_WIDTH+2], next_word_count, 2'b00};
        end
        S_FILL: begin
                mem_cs_o <= 1;
                mem_we_o <= 0;
                mem_addr_o <= {addr_rw[31:LINE_WORDS_WIDTH+2],
next_word_count, 2'b00};
        end
endcase
```

Simulation (1)

```
module inst (
        input wire clk,
        input wire rst,
        input wire [3:0] index,  // instruction index
        output wire valid,  // stop running if valid is 0
        output wire write,  // write enable signal for cache
        output wire [31:0] addr  // address for cache
        );
        reg [33:0] data [0:7];
        initial begin  // clock cycles are only for reference
                data[0] = 34'h200000004;  // read miss              1+17
                data[1] = 34'h300000018;  // write miss             1+17
                data[2] = 34'h200000008;  // read hit               1
                data[3] = 34'h300000014;  // write hit              1
                data[4] = 34'h210000004;  // read & clean replace   1+17
                data[5] = 34'h310000018;  // write & dirty replace  1+17*2
                data[6] = 34'h310000008;  // write hit              1
                data[7] = 34'h0;          // end                    total: 92
        end
        assign
                valid = data[index][33],
                write = data[index][32],
                addr = data[index][31:0];
endmodule
```

Simulation (2)

```
`timescale 1ns / 1ps

module top (
	input wire clk,
	input wire rst,
	output reg [7:0] clk_count = 0,
	output reg [7:0] inst_count = 0,
	output reg [7:0] hit_count = 0
	);

	// instruction
	reg [3:0] index = 0;
	wire valid;
	wire write;
	wire [31:0] addr;
	wire stall;
	inst INST (
		.clk(clk),
		.rst(rst),
		.index(index),
		.valid(valid),
		.write(write),
		.addr(addr)
	);
```

```
always @(posedge clk) begin
	if (rst)
		index <= 0;
	else if (valid && ~stall)
		index <= index + 1'h1;
end
// ram
wire mem_cs;
wire mem_we;
wire [31:0] mem_addr;
wire [31:0] mem_din;
wire [31:0] mem_dout;
wire mem_ack;
data_ram #(
	.ADDR_WIDTH(5),
	.CLK_DELAY(3)
	) RAM (
	.clk(clk),
	.rst(rst),
	.addr({26'b0, mem_addr[5:0]}),
	.cs(mem_cs),
	.we(mem_we),
	.din(mem_din),
	.dout(mem_dout),
	.stall(),
	.ack(mem_ack)
	);
```

Simulation (3)

```
// cache
cmu CMU (
        .clk(clk),
        .rst(rst),
        .addr_rw(addr),
        .en_r(~write),
        .data_r(),
        .en_w(write),
        .data_w({16'h5678, clk_count, inst_count}),
        .stall(stall),
        .mem_cs_o(mem_cs),
        .mem_we_o(mem_we),
        .mem_addr_o(mem_addr),
        .mem_data_i(mem_dout),
        .mem_data_o(mem_din),
        .mem_ack_i(mem_ack)
);
```

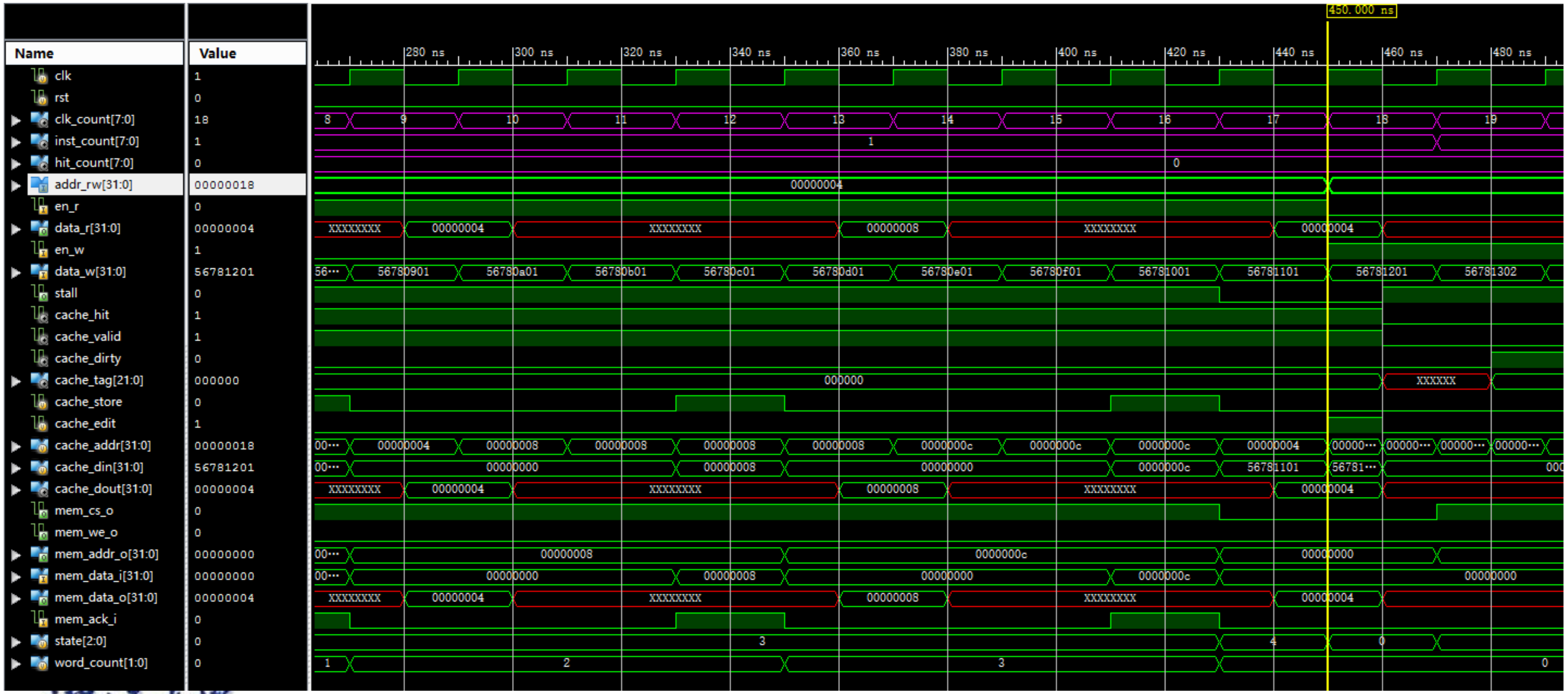
```
// counter
reg stall_prev;

always @(posedge clk) begin
        if (rst)
                stall_prev <= 0;
        else
                stall_prev <= stall;
end

always @(posedge clk) begin
        if (rst) begin
                clk_count <= 0;   // 时钟计数
                inst_count <= 0;  // 指令计数
                hit_count <= 0;   // 命中计数
        end
        else if (valid) begin
                clk_count <= clk_count + 1'h1;
                inst_count <= index + 1'h1;
                if (~stall_prev && ~stall)
                        hit_count <= hit_count + 1'h1;
        end
end

endmodule
```

Simulation (4)

```
module sim_top;

    // Inputs
    reg clk;
    reg rst;

    // Outputs
    wire [7:0] clk_count;
    wire [7:0] inst_count;
    wire [7:0] hit_count;

    // Instantiate the Unit Under Test (UUT)
    top uut (
        .clk(clk),
        .rst(rst),
        .clk_count(clk_count),
        .inst_count(inst_count),
        .hit_count(hit_count)
    );

    initial begin
        // Initialize Inputs
        clk = 0;
        rst = 1;

        // Wait 100 ns for global reset to finish
        #95 rst = 0;

        // Add stimulus here


    end

    initial forever #10 clk = ~clk;

endmodule
```

Result (1)

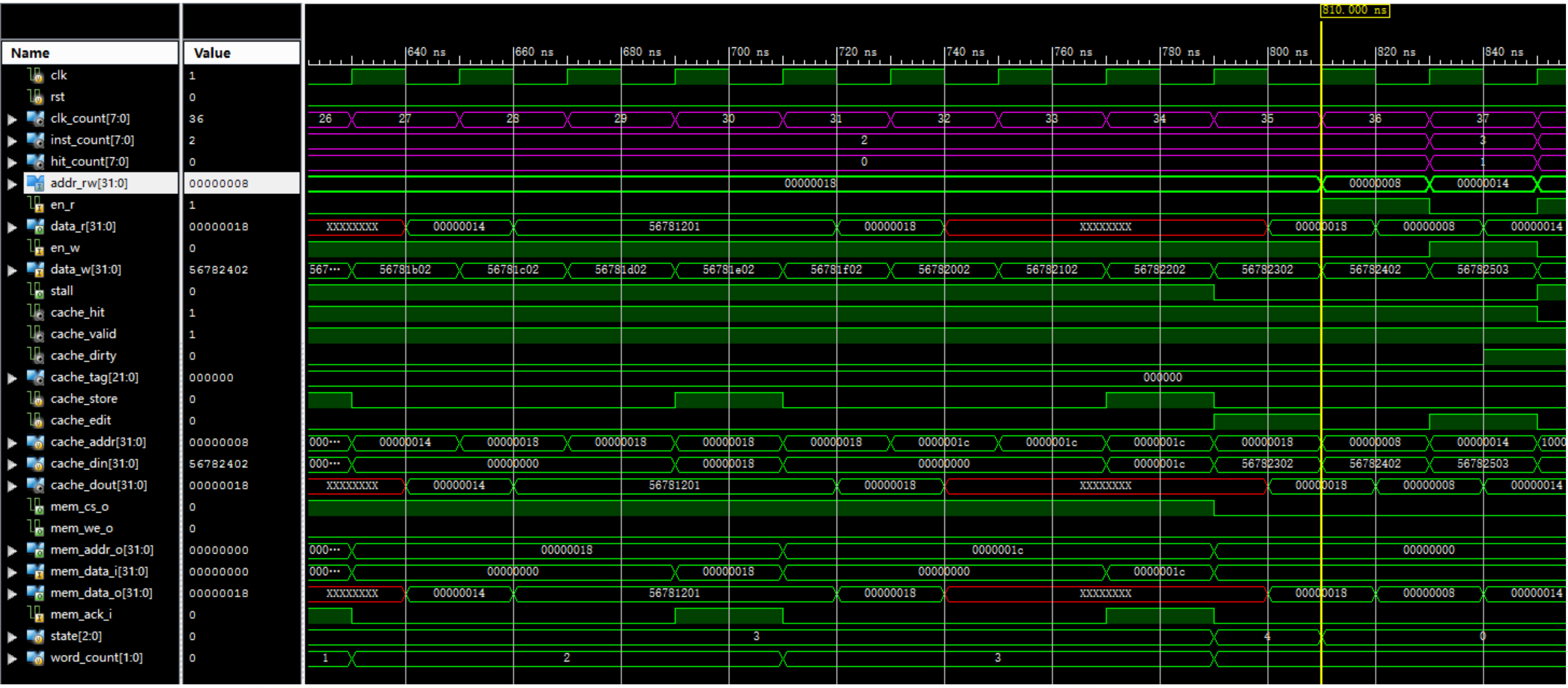
- **Read miss at 0x00000004**



Name	Value
clk	1
rst	0
clk_count[7:0]	18
inst_count[7:0]	1
hit_count[7:0]	0
addr_rw[31:0]	00000018
en_r	0
data_r[31:0]	00000004
en_w	1
data_w[31:0]	56781201
stall	0
cache_hit	1
cache_valid	1
cache_dirty	0
cache_tag[21:0]	000000
cache_store	0
cache_edit	1
cache_addr[31:0]	00000018
cache_din[31:0]	56781201
cache_dout[31:0]	00000004
mem_cs_o	0
mem_we_o	0
mem_addr_o[31:0]	00000000
mem_data_i[31:0]	00000000
mem_data_o[31:0]	00000004
mem_ack_i	0
state[2:0]	0
word_count[1:0]	0

Result (2)

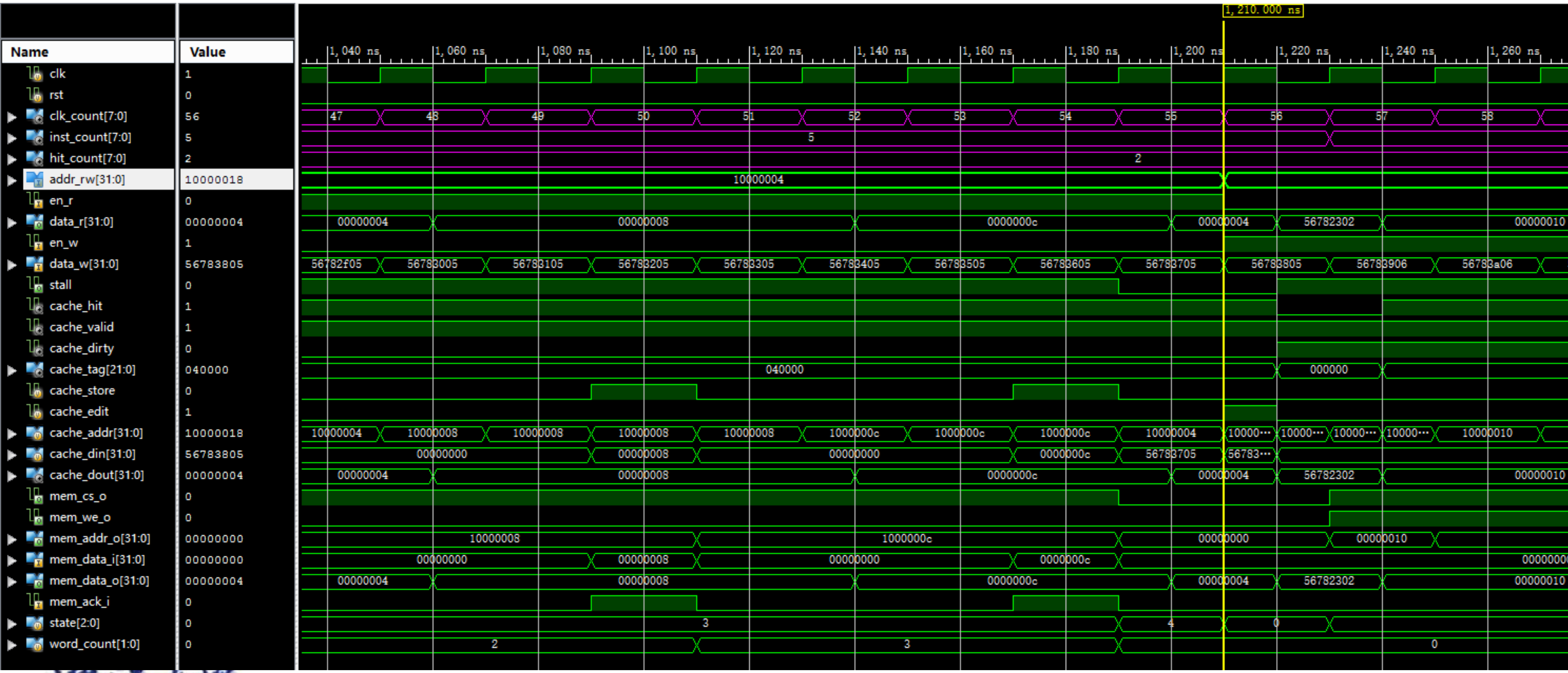
- Write miss at 0x00000018
- Read hit at 0x00000008
- Write hit at 0x00000014



Name	Value
clk	1
rst	0
clk_count[7:0]	36
inst_count[7:0]	2
hit_count[7:0]	0
addr_rw[31:0]	00000008
en_r	1
data_r[31:0]	00000018
en_w	0
data_w[31:0]	56782402
stall	0
cache_hit	1
cache_valid	1
cache_dirty	0
cache_tag[21:0]	000000
cache_store	0
cache_edit	0
cache_addr[31:0]	00000008
cache_din[31:0]	56782402
cache_dout[31:0]	00000018
mem_cs_o	0
mem_we_o	0
mem_addr_o[31:0]	00000000
mem_data_i[31:0]	00000000
mem_data_o[31:0]	00000018
mem_ack_i	0
state[2:0]	0
word_count[1:0]	0

Result (3)

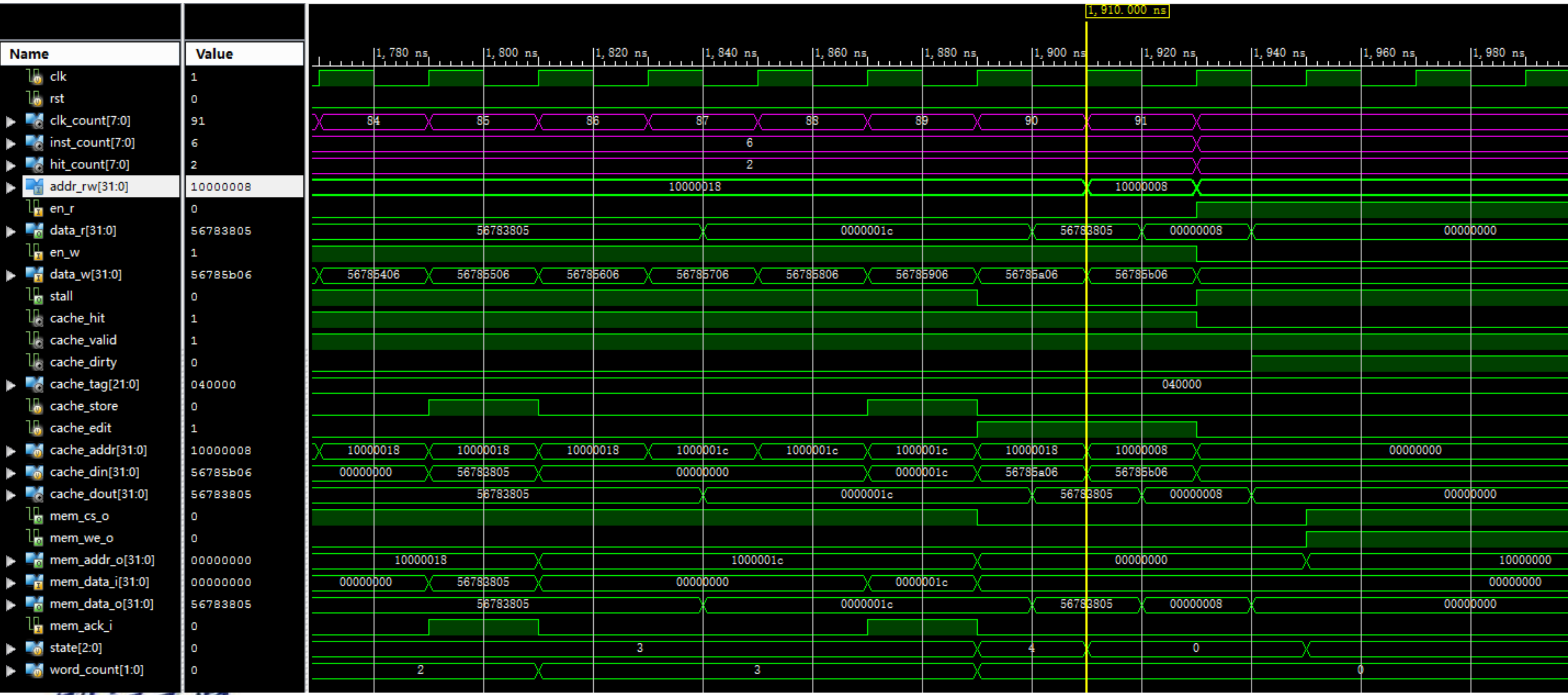
- **Read & clean replace at 0x10000004**



Name	Value
clk	1
rst	0
clk_count[7:0]	56
inst_count[7:0]	5
hit_count[7:0]	2
addr_rw[31:0]	10000018
en_r	0
data_r[31:0]	00000004
en_w	1
data_w[31:0]	56783805
stall	0
cache_hit	1
cache_valid	1
cache_dirty	0
cache_tag[21:0]	040000
cache_store	0
cache_edit	1
cache_addr[31:0]	10000018
cache_din[31:0]	56783805
cache_dout[31:0]	00000004
mem_cs_o	0
mem_we_o	0
mem_addr_o[31:0]	00000000
mem_data_i[31:0]	00000000
mem_data_o[31:0]	00000004
mem_ack_i	0
state[2:0]	0
word_count[1:0]	0

Result (4)

- **Write & dirty replace at 0x10000018**
- **Write hit at 0x10000008**



Name	Value
clk	1
rst	0
clk_count[7:0]	91
inst_count[7:0]	6
hit_count[7:0]	2
addr_rw[31:0]	10000008
en_r	0
data_r[31:0]	56783805
en_w	1
data_w[31:0]	56785b06
stall	0
cache_hit	1
cache_valid	1
cache_dirty	0
cache_tag[21:0]	040000
cache_store	0
cache_edit	1
cache_addr[31:0]	10000008
cache_din[31:0]	56785b06
cache_dout[31:0]	56783805
mem_cs_o	0
mem_we_o	0
mem_addr_o[31:0]	00000000
mem_data_i[31:0]	00000000
mem_data_o[31:0]	56783805
mem_ack_i	0
state[2:0]	0
word_count[1:0]	0

Performance Analysis

- **Miss Penalty**
  - When clean, MP = 17
  - When dirty, MP = 34
- **Result**
  - CLK_COUNT = 92
  - INST_COUNT = 7
  - HIT_COUNT = 3

Checkpoints

- **CP1:**

  Waveform Simulation of Cache Line.

- **CP2:**

  Waveform Simulation of CMU.

Thanks!